第一章　Macintosh 系统概述

1.1　系统软件介绍

Macintosh的系统软件包括操作系统、工具箱（Toolbox）以及其它系统软件，如图1-1所示。

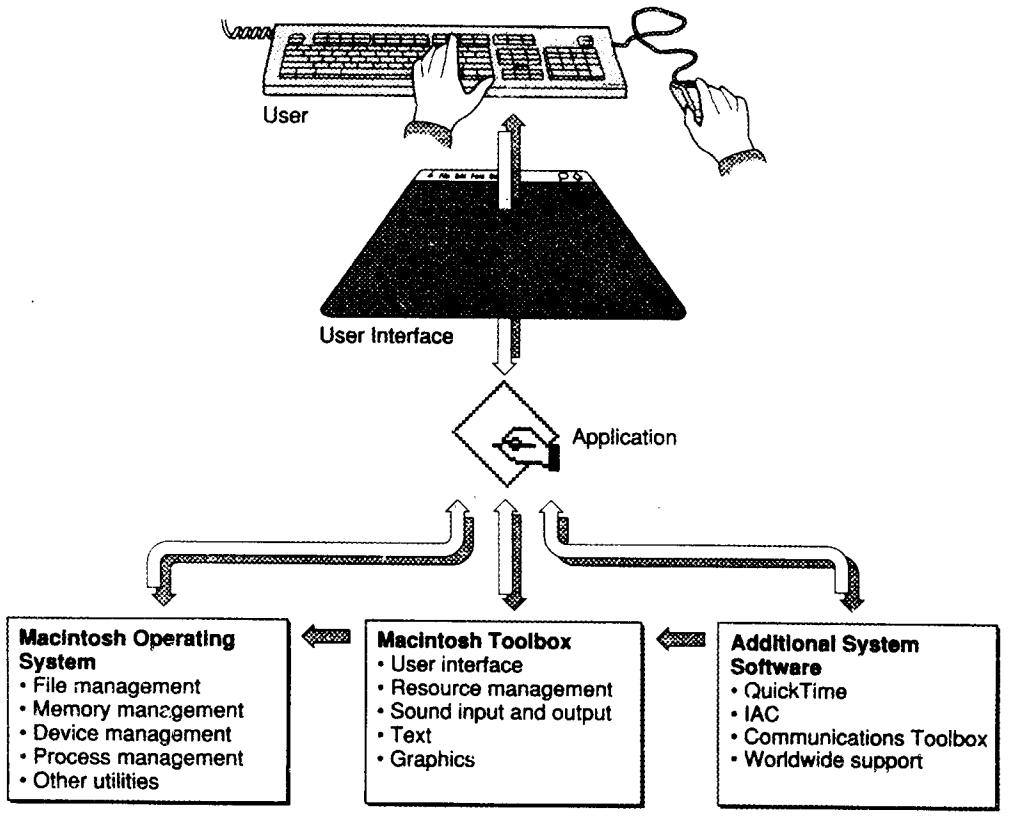

图 1-1　Macintosh 系统软件概貌

Macintosh 的一个突出特点是具有良好的用户界面，而这一点正是得益于它丰富的系统软件。Macintosh的系统软件中目前有几千个过程可供调用。这些过程在逻辑上被分为若干功能组，用来处理不同的任务或用户接口，我们称之为管理器（managers）。如，窗口管理器（Window Manager）可用来建立、移动、隐藏窗口以及改变窗口大小等。

Macintosh图形用户界面中有一个重要部分，叫做Finder，实际上它不是系统软件的一部分，只是一个应用程序。Finder负责跟踪文件并管理用户的桌面。

下面我们分别介绍Macintosh的工具箱、操作系统及其它系统软件。

1.1.1 Macintosh工具箱

Macintosh工具箱提供一系列过程，应用程序可以调用这些过程来实现各种用户界面。工具箱保证了用户界面的方便和一致性，也减小了应用程序的规模，缩短了开发时间，而且使应用程序更易于维护。表1-1中给出了工具箱中常用的管理器。

工具箱中的一个重要部分是QuickDraw，它用于处理绘图和其它图形操作。由于Macintosh用户界面在很大程度上是一个图形界面，因此实际上工具箱中所有其它管理器都要调用QuickDraw中的过程。如，窗口管理器要调用QuickDraw绘制窗口边框及其它需要的部分。所以编程时，在初始化工具箱的其它管理器之前，首先要初始化QuickDraw。

1.1.2 Macintosh操作系统

Macintosh操作系统提供完成基本的低层任务的过程，如，文件的输入和输出、内存管理、进程和设备控制。工具箱是在操作系统之上的，它提供用于实现标准用户界面的过程。工具箱需要调用操作系统来完成低层操作，应用程序也可以直接调用操用系统。

工具箱是应用程序与用户间的媒介，而操作系统则是应用程序与系统硬件间的媒介。文件管理器和设备管理器将应用程序与数据存储的硬件细节隔离开。同样，内存管理器帮助应用程序在其逻辑地址空间中分配和释放内存。内存管理器负责将逻辑地址空间映射到RAM的物理地址空间，并在必要时移动可重定位内存块以产生需要的新块。表1-2列出了Macintosh操作系统的主要部分。

1.1.3 其它系统软件

有一些其它软件，既不属于工具箱，也不属于操作系统。这些系统软件提供一系列强有力的服务，如处理文本，以及支持各种不同语言的文本处理。这些软件中还包括应用程序间的通讯机制、QuickTime、以及通讯工具箱。

最后，还有一点要说明的就是操作系统和工具箱所在的位置。Macintosh操作系统的一部分存放于ROM中，另外一些以系统文件的方式存放于启动磁盘上。工具箱则全部在ROM中，这一点是Macintosh的独到之处。

表 1-1　Macintosh 工具箱

管理器	说　明
QuickDraw	完成所有屏幕显示操作，包括描绘所有图形及文字。
窗口管理器 (Window Manager)	创建及管理各种窗口
对话管理器 (Dialog Manager)	创建及管理对话框，即一种特殊窗口。对话框一般用来警告用户有异常情况，或者请用户输入信息。
控制管理器 (Control Manager)	建立及管理控制部件(controls)，如按钮(buttons)，选项按钮(radio buttons)，检查框(checkboxs)，弹出式菜单(pop-up menus)，滚动条(scroll bars)，以及应用程序定义的控制部件。
菜单管理器 (Menu Manager)	建立及管理应用程序的菜单条及所含的菜单，并且处理菜单的绘制及用户在菜单中的操作。
事件管理器 (Event Manager)	向应用程序报告事件，如描述用户动作的事件或改变进程状态的事件。并协助应用程序间通讯。
文本编辑 (Text Edit)	提供简单的文本格式及文本编辑功能，如文本输入、选取、剪、贴。不以处理文本为主的应用程序可以用这些功能完成大多数文本操作。
资源管理器 (Resource Manager)	使应用程序可以读、写资源。应用程序中用到的任何静态数据（如菜单、光标及窗口）都可有效地存储为资源，应用程序可以定义自己的资源。
Finder界面	使应用程序可以与Finder交互。
剪贴管理器 (Scrap Manager)	支持应用程序之间的信息剪贴。
标准文件管理包 (Standard File Package)	提供标准对话框，使用户可以选择一个文件打开，或给需存储的文件定位并命名。
帮助管理器 (Help Manager)	使应用程序可以提供联机帮助
列表管理器 (List Manager)	使应用程序可以建立列表。
声音管理器 (Sound Manager)	提供声音输出能力。
声音输入管理器 (Sound Input Manager)	为配有声音输入设备的Macintosh计算机提供声音输入能力。

表 1-2　Macintosh 操作系统

管理器	说　明
进程管理器 (Process Manager)	处理启动、调度以及终止应用程序。并提供被打开进程的信息
内存管理器 (Memory Manager)	管理应用程序分区中内存的动态分配和释放。
虚拟内存管理器 (Virtual Memory Manager)	提供虚存服务（使逻辑地址空间大于实际可用的RAM空间）。
文件管理器 (File Manager)	提供对文件系统的访问，允许应用程序建立、打开、读、写和关闭文件。
替身管理器 (Alias Manager)	帮助寻找指定的文件、目录、或卷。
磁盘初始化管理器 (Disk Initialization Manager)	管理磁盘的初始化
SCSI管理器	控制Macintosh主机与连接在SCSI口上的外部设备间的信息交换。
时间管理器 (Time Manager)	使一个过程可以周期地执行，或延迟一段指定的时间后再运行。
垂直重画管理器 (Vertical Retrac manager)	使得在执行一个过程时，可以同步重画屏幕。
关机管理器 (Shut down Manager)	使你可以在计算机关机或重启动时执行一个过程。

1.2 内存管理介绍

1.2.1 内存分区

在Macintosh多任务环境下，同时可以打开多个应用程序。这时，内存的一部分由操作系统使用，剩下的部分由打开的应用程序共用。在操作系统启动时，将内存分为两部分，一部分是系统分区，从地址0开始向上延伸。另一部分分配给打开的应用程序或其它软件。内存的组织如图1-2所示，以下分别说明。

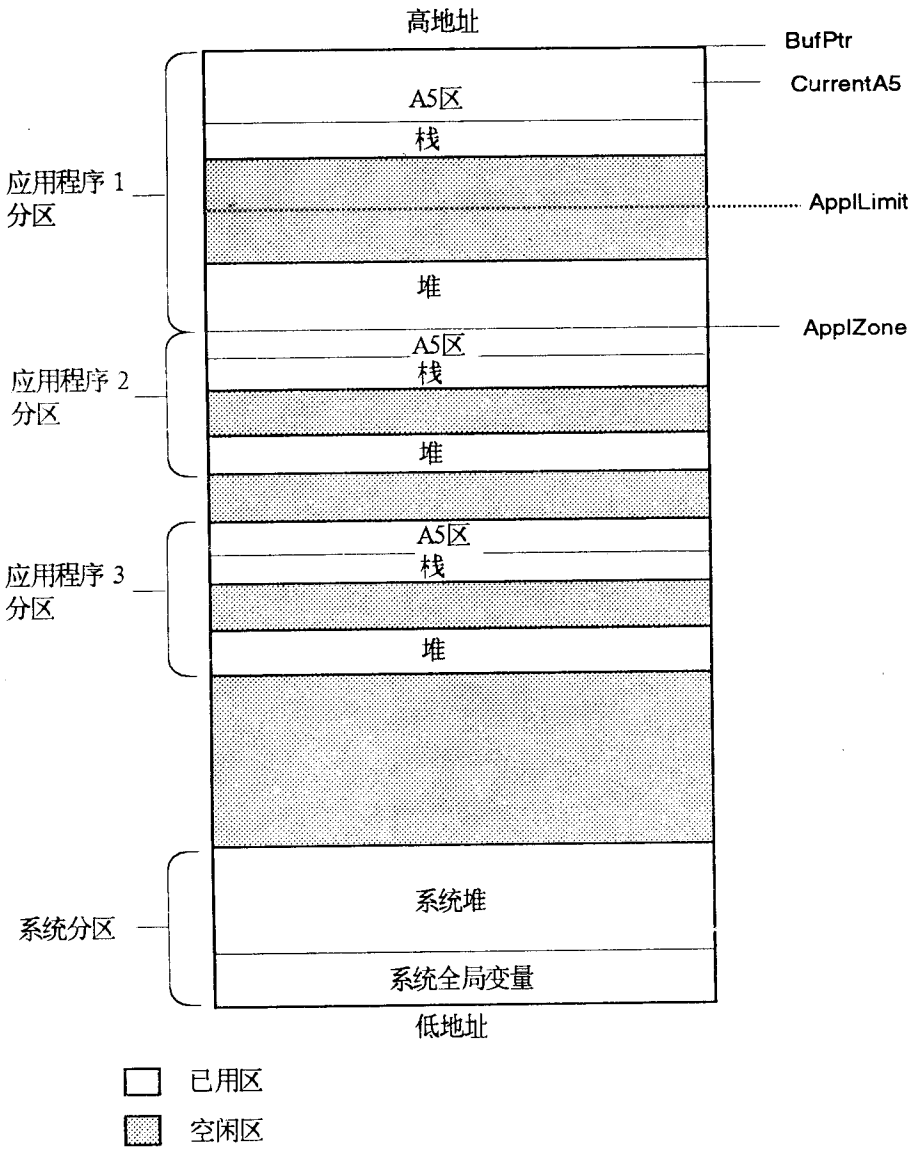

图1-2 内存组织

一、系统分区

1. 系统堆

系统分区的主要部分是系统堆。一般来说，系统堆由操作系统和其它系统软件用来存放如系统资源、系统代码段以及系统数据结构等内容。所有的系统缓冲区都安排在系统堆中。

2. 系统全局变量

最低内存分配给系统全局变量，操作系统用这些变量来保存操作系统环境的各种信息，以及当前应用程序的有关信息。

一般请不要读、写系统全局变量，这些变量多数是没有文档的，其值的改变也是不可预知的，通常，对应用程序有用的系统全局变量，其值都可由系统软件提供的过程来读写。

二、应用程序分区

当一个应用程序被启动时，操作系统为其分配一个分区，称应用程序区，其中包含必要的应用程序代码段以及其它与应用程序有关的数据。图 1-3 是应用程序区的示意图。

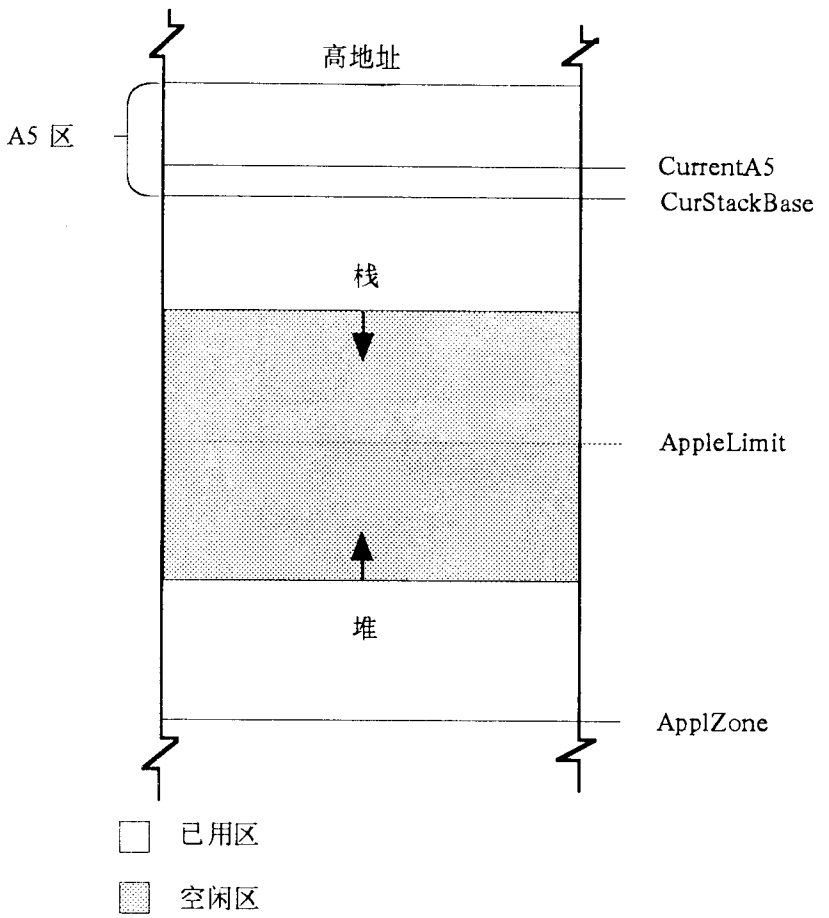

图 1-3　应用程序区

应用程序区分为三个主要部分:

- 应用程序栈
- 应用程序堆
- 应用程序全局变量及A5区域

应用程序堆位于应用程序区的低地址端，在需要的时候向上延伸。A5区域具有固定大小，位于高地址端。栈从A5区域的低地址端开始，向下延伸。

在图1-3中可看到，栈和堆之间有一段未用区域。这一区域为栈的生长提供了空间，以免侵占堆空间。若侵占了堆空间，堆中的数据将被破坏。

全局变量ApplLimit标志着堆生长的上限，但栈的生长并不受此限制。因此如果应用程序中用了过多的嵌套过程，过多的局部变量，或大量使用递归，都会使栈的生长超过界限。由于使用局部变量和递归并不调用内存管理器来分配栈空间，因而内存管理器无法防止这种情况。但是，操作系统大约每秒查看60次，检查栈是否侵入到堆中，如果是，则“栈探测器”产生一个系统错误。

下面分别介绍三个部分的作用

1. 应用程序栈

由于栈的后进先出特性，在执行函数和过程调用时非常有用。当应用程序调用一个过程时，系统便在应用程序栈中自动分配一个栈框架（stack frame）。栈框架中包括子过程的参数、局部变量和返回地址。

2. 应用程序堆

应用程序堆中的空间是根据需要动态分配和释放的。除栈中的项目外，应用程序的所有项目都在堆中。如，代码段和已装入内存的资源。堆中还有其它动态分配的项目，如窗口记录、对话记录、文档数据等等。

你可以直接或间接地调用内存管理器分配应用程序堆中的内存空间。例如，调用NewHandle函数，或调用NewWindow，NewWindow又将调用内存管理器中的过程。堆中的内存空间是按块分配的。块可以是任意大小。

在分配和释放堆中的内存块时，内存管理器负责所有的管理以跟踪内存块。分配和释放操作完全是无序的，因此，应用程序运行一段时间后，堆可能会被分裂成已分配块和未分配块的混合，如图1-4所示。这种分裂称为堆分裂。

分裂的结果是，某一时刻，内存管理器再也无法满足应用程序对新的内存块的请求，虽然此时空闲空间的总容量可能超过应用程序需要的块大小，但空闲空间已被划分成很小的碎片。在这种情况下，内存管理器就要移动已分配块，使之集中，将空闲块收集成一个大的块。这种操作称堆压缩。

只要在堆压缩期间已分配块可以自由移动，堆分裂就不成为问题。但是，在两种情况下块是不能自由移动的，一是遇到不可重定位块，二是可重定位块被暂时锁在某一位置。这个问题将在稍后讨论。

3. 应用程序全局变量和A5区域

应用程序的全局变量存放在靠近应用程序区顶端的A5区域中。A5区域中包括4种数据:

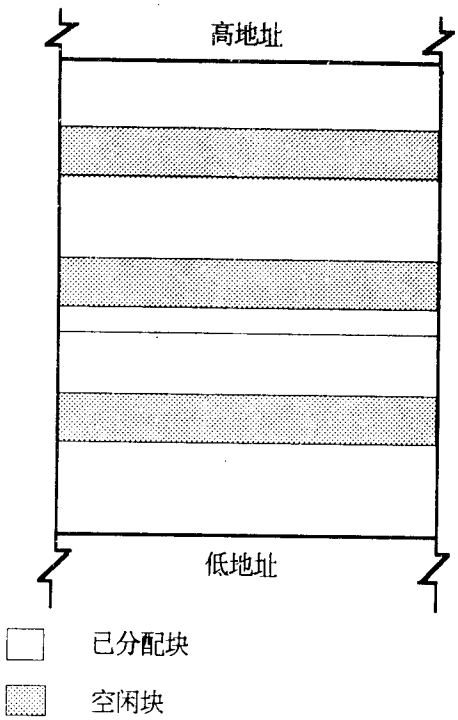

图 1-4　分裂后的堆

■应用程序全局变量
■应用程序 QuickDraw 全局变量
■应用程序参数
■应用程序跳转表

虽然不同应用程序的全局变量和跳转表所占空间大小不同，但对一个应用程序来说，上述每一项都是固定大小的。图 1-5 给出了 A5 区域的一般组织形式。

系统全局变量 CurrentA5 指向当前应用程序的全局变量和应用程序参数的分界处。这个分界很重要，操作系统正是用它来访问应用程序全局变量、QuickDraw 全局变量、应用程序参数以及跳转表的。将这些信息总称为 A5 区域，是因为操作系统用寄存器 A5 指向这一分界。

应用程序的 QuickDraw 全局变量包括有关绘图环境的信息。

应用程序跳转表中包括应用程序中每个可调用过程的入口。

应用程序参数是位于应用程序全局变量之上的 32 字节，这部分空间是保留给操作系统使用的。其中的第一个长字是一个指向 QuickDraw 全局变量的指针。

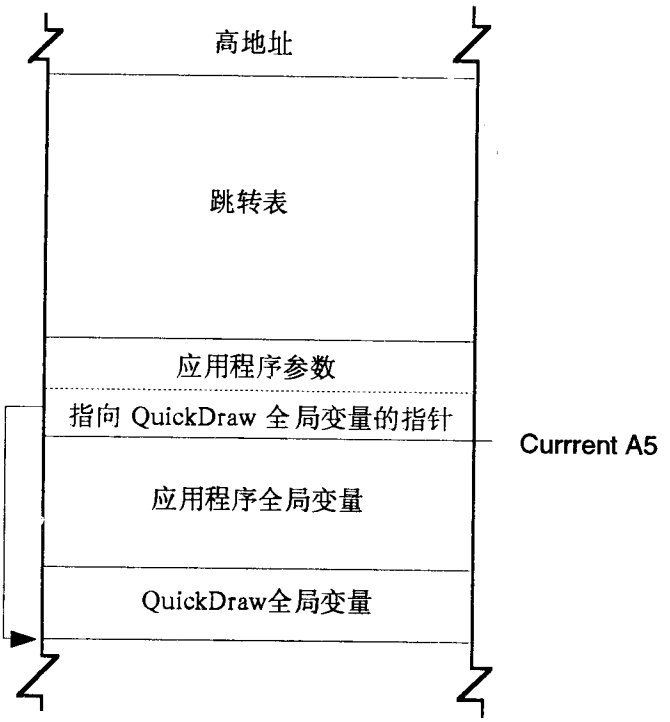

图 1-5　应用程序A5区域的组织

1.2.2　内存块

内存管理器在应用程序堆中分配的块有两种：可重定位块和不可重定位块。

不可重定位块在堆中的地址是固定的，用Ptr（指针）型变量来引用。可重定位块可以在堆内移动，用Handle型（二级指针）变量来引用。

应用程序可以用Hlock过程给可重定位块加锁，一但加锁该块便不能移动了。然后，可用HUnlock过程解锁，使其重又可移动。

当应用程序的进程状态改变或将要改变时，事件管理器向应用程序送一个操作系统事件。例如，用户将一个应用程序带到前台，进程管理器便通过事件管理器发送一个事件给应用程序。

当其它应用程序或进程直接与你的应用程序通讯时，事件管理器便向你的应用程序发送一个高层事件。

事件管理器为每一个打开的应用程序保持一个事件流。应用程序一般是取出一个事件，处理后，再取下一个事件，处理它，等等，如此循环。只有当用户选择退出应用程序时，才结束这种循环。

当你的应用程序一开始启动时，在接收和处理事件之间，必须首先进行初始化。如，初始化工具箱中的管理器，建立应用程序菜单，以及其它初始化。初始化之后，应用程序进入事件循环，调用工具箱中的Wait Next Event函数从系统获取信息。应用程序的一般流程如图1-6所示。

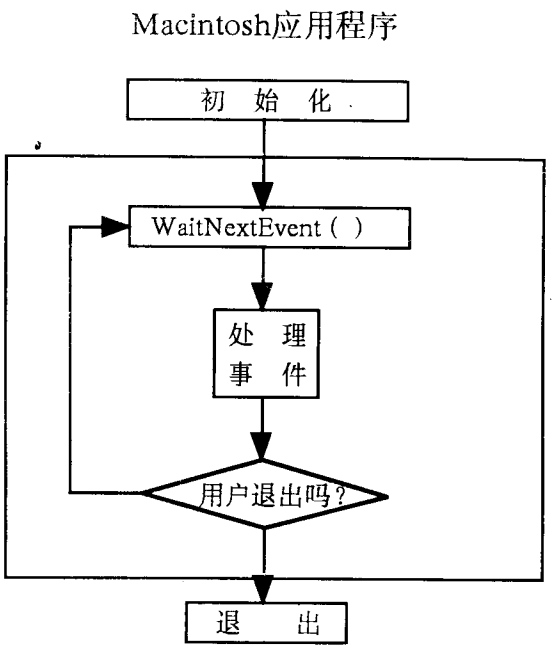

图 1-6 Macintosh 应用程序的一般流程。

1.3 事件的概念

一个好的 Macintosh 应用程序，其突出特点就是由用户控制。即用户可以在任何时间进行任何操作，包括打开应用程序菜单，选择一个菜单命令，键入一些字符，移动某个窗口等等。一个关键的概念是，必须使用户感觉你的应用程序随时都准备为他服务。

即使你的应用程序正在忙于进行某种费时的操作（例如，将文档存入磁盘），不允许用户做别的事，你也应为用户提供一种安全的方式来终止这一操作。一般来说，应用程序可以显示一个对话框，提示出户现在正在进行某种费时的操作，并指出终止这一操作的安全方法。

这种以用户为中心的设计，其基础是："事件驱动编程模式"（event-driven programming model）。换句话说，就是系统软件将用户的操作分成一些事件，一个一个地送给应用程序处理。例如，当用户按下一个键时，系统软件将有关这一事件的信息送给应用程序。这些信息包括：按的是哪个键、什么时间按下的，按键的同时是否有某个修改键（如，command键）一起按下，等等。应用程序执行相应的动作以响应这一事件。

应用程序可以接收很多种事件。事件通常被分为三类：

- 低层事件（low-evel events）
- 操作系统事件（operating-system events）
- 高层事件（high-level events）

当用户按下鼠标键，释放鼠标键，按下键盘键，或插入磁盘时，事件管理器便送一个

低层事件给应用程序，当应用程序需要激活一个窗口，或修改窗口时，事件管理器也回送一个低层事件。若应用程序请求事件时没有任何事件，事件管理器便返回一个空事件（null event）。

1.4 进程控制

一个进程就是一个打开的应用程序或桌面附件。

1.4.1 协作多任务环境

Macintosh操作系统，Finder，以及其它系统软件一起提供了一个多任务环境。用户可以在其中同时打开多个应用程序，并在应用程序间任意切换。要在这一环境中运行，你的应用程序必须遵循一定的原则。

例如，你的应用程序应该包括SIZE资源，它说明应用程序启动时需要分配多大的内存区。而且，你的应用程序还应该定期发出事件请求，以使操作系统有机会调度其它应用程序执行。由于所有应用程序的协调运行取决于相互间的协作，这一环境便称为协作多任务环境。

在多任务环境中，应用程序可以把运行时间较长的任务放在后台，使用户可以把其它应用程序放到前台，由前台应用程序决定在什么时候将多少时间交给后台应用程序。只有在前台进程发出事件请求（如，调用WaitNextEvent），并且又没有事件等待前台进程处理时，后台进程才能得到运行时间。这时进程管理器向后台进程发送一个空事件，通知它现在是当前进程，可以进行需要的后台处理。

WaitNextEvent过程定义如下：

```
Boolean   WaitNextEvent ( Short  event_mask,  EventRecord  *the_event,
                          long  sleep_time,  RgnHandle  mouse_rgn )
```

前台应用程序在WaitNextEvent调用中，用sleep_time参数来规定它想给出多少时间（以1/60秒为单位）。

如果前台应用程序通过sleep_time传送一个0值，将使后台程序不能正常工作。前台应用程序应该给出尽可能多的时间，这个时间取决于应用程序对时间的需要。以本书中的示例程序Target2为例（见附录），它有文本编辑功能，在编辑文本时，需要闪动文本插入符“|”。通常控制板中最快的插入符闪动速度为每秒5次。这意味着如果Target2需要闪动文本插入符的话，它必须每秒得到5次控制，两次闪动之间约有12/60秒间隔。于是，当Target2在前台调用WaitNextEvent时便可将12传送给参数sleep_time。

同样，当后台应用程序获得控制后，也应尽快通过下一个WaitNextEvent调用来归还控制，以避免超过前台程序sleep_time参数规定的时限。

1.4.2 应用程序间的切换

为了便于应用程序间的切换，系统提供了两个事件："suspend"和"resume"。

当进程管理器要将一个应用程序从前台切换到后台（简称“换出”）时，进程管理器便向应用程序发送一个suspend事件。这只是一个信号，通知应用程序准备换出。这时进程管理器并不将应用程序立即换出，而是给它一个机会来处理suspend事件。应用程序将在发出下一个事件请求时被换出。

同样，对于将切换到前台（简称“换入”）的应用程序，进程管理器将发送一个resume事件作为信号。

当应用程序收到suspent事件时，建议做以下操作：

1. 使活动窗口成为非活动窗口。
2. 可后台运行的程序，应设置一个应用程序全局变量来反映它正处于后台。
3. 具有私有剪贴数据的应用程序，必须使用剪贴管理器，将其私有剪贴数据转换成公用剪贴数据。

“剪贴数据”是指用户在“剪”或“复制”操作中选择的数据。系统规定了公共的“剪贴数据”格式，以便调用剪贴管理器在不同应用程序之间剪、贴、复制数据。

可是有些应用程序，如数据库应用程序，其数据格式比较复杂或比较长。当这些应用程序在前台运行时，可能宁愿存在私有存储块中，以自定格式保存自己的剪贴数据，因为这样其数据复制和贴的速度会快些。

只要应用程序一直在前台运行，其私有剪贴数据总是好用的。而一但要将其换出时，就必须将私有剪贴数据转换成公用的，才能贴入被换入的应用程序中。

当一个应用程序接到resume事件时，建议做以下操作：

1. 如果被换入的应用程序有一个前端窗口，应使其成为活动窗口。
2. 可后台运行的应用程序，应设置一个应用程序全局变量来反映它正处于前台。
3. 如果应用程序要保持它自己的私有剪贴数据，它应调用剪贴管理器将其公用剪贴数据转换成私有的。

应用程序间的切换有两种：主切换（major switch）和次切换（minor switch）。

主切换是一次完全切换，涉及到两个应用程序，一个被换到前台，一个被换到后台。系统重新设置环境寄存器和全局变量，变更活动窗口，将得到前台时间的应用程序的窗口移到前端，将控制交给前台应用程序，并接收一个resume事件。

当一个应用程序在后台得到时间时，便发生一个次切换。在次切换期间，设置环境寄存器和全局变量，并将控制交给获得后台时间的应用程序。与主切换不同的是，不将应用程序窗口移到前端，应用程序也不接收resume事件。

一个应用程序并不是可以自动使用suspend和resume事件的，你必须在其SIZE资源中指定这一能力。应用程序有很多预置项都可以通过SIZE资源传递给系统。图1-7所示为ResEdit中看到的应用程序Target2的SIZE资源中的一部分。

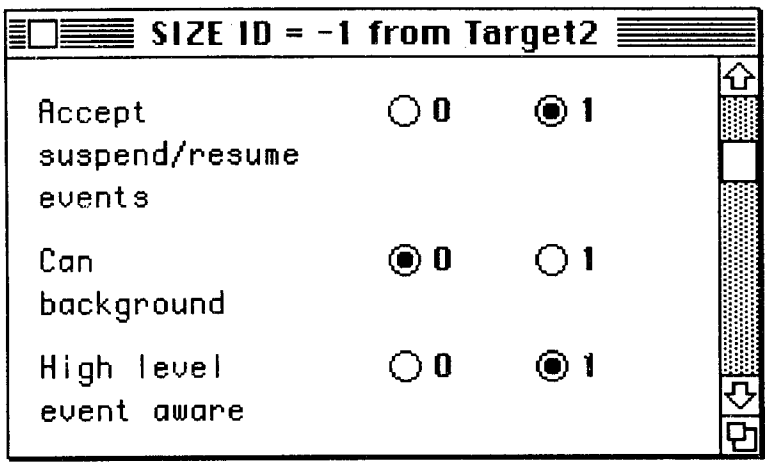

图 1-7　SIZE 资源示例

1.5　文件系统介绍

本节简单介绍 Macintosh 的文件系统，包括文件的基本结构和文件系统的层次结构（HFS）。并介绍如何通过文件说明来引用文件，以及文件替身的概念。

1.5.1　文件的基本结构

Macintosh文件按其不同用途可分为若干种类。常用的有文档（document）和应用程序（application）。一般来说，文档是用户可以建立、编辑的文件，可以用两次点按其图标的方式打开。应用程序包括可执行代码及其专用资源（resource）和数据（data）。

如图1-8所示，每个Macintosh文件都分为两个分支（forks），称为数据分支和资源分支。

资源分支中包含文件的资源。如果文件是一个应用程序，其资源分支一般包括用于描述应用程序窗口、菜单、对话框、图标的资源，以及可执行代码本身。应用程序的SIZE资源是一个很重要的资源，其中指定应用程序的一些能力及其运行时的内存要求。如果文件是一个文档，其资源分支一般包括预置信息、窗口位置以及文档的字体、图标等等。

数据分支中包含文件的数据。简单地说，就是连续的数据字节，不具有任何内部结构（资源分支中的数据是有结构的）。而你的应用程序应负责正确解释这些字节。例如，一个文档文件的数据分支可能包括一封信的文字。

将数据存入数据分支还是资源分支，在很大程序上取决于是否能将数据有效地结构化以作为资源。例如，你要存一些姓名和电话号码，你很容易设计一个资源类型，将每对姓名和电话号码作为一个资源。要取出资源数据，只需指定资源类型和资源号（ID），而无需知

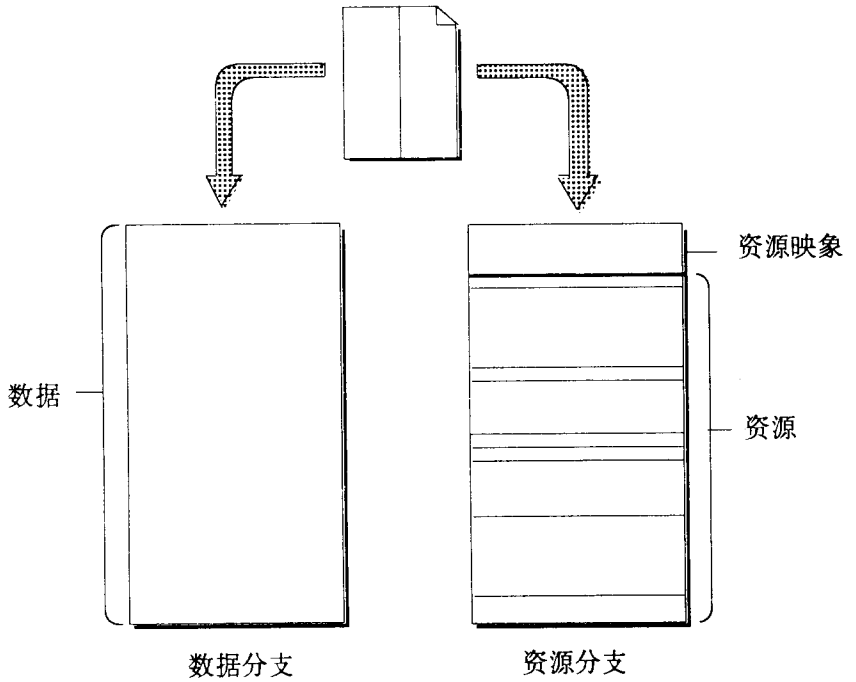

图 1-8　Macintosh 文件的两个分支

道它有多少字节。但是在有些情况下，数据难以放入资源管理器要求的结构中。例如，一篇文章，其长度通长是不定的，因此，存入数据分支中比较合适。

1.5.2　文件系统的层次结构

Macintosh 操作系统组织文件的方式称作层次文件系统（hierarch file system，简称 HFS）。在 HFS 中，文件被分组存于不同的目录中，目录称为文件夹（folder），而若干个目录又可组成上层目录，如图 1-9 所示。Finder 与文件管理器一起负责管理文件及文件夹。

1.5.3　文件说明和文件替身

一、文件说明

虽然我们可以通过文件名和工作目录号（working directory ID）来引用文件的方法。但是，使用工作目录号的方法有一些问题。

首先，工作目录是动态指定的，因此文件每次打开时可能具有不同的工作目录号。这意味着文件关闭后，工作目录号就没有意义了。第二，系统对于一次可能打开的工作目录数是有限制的。

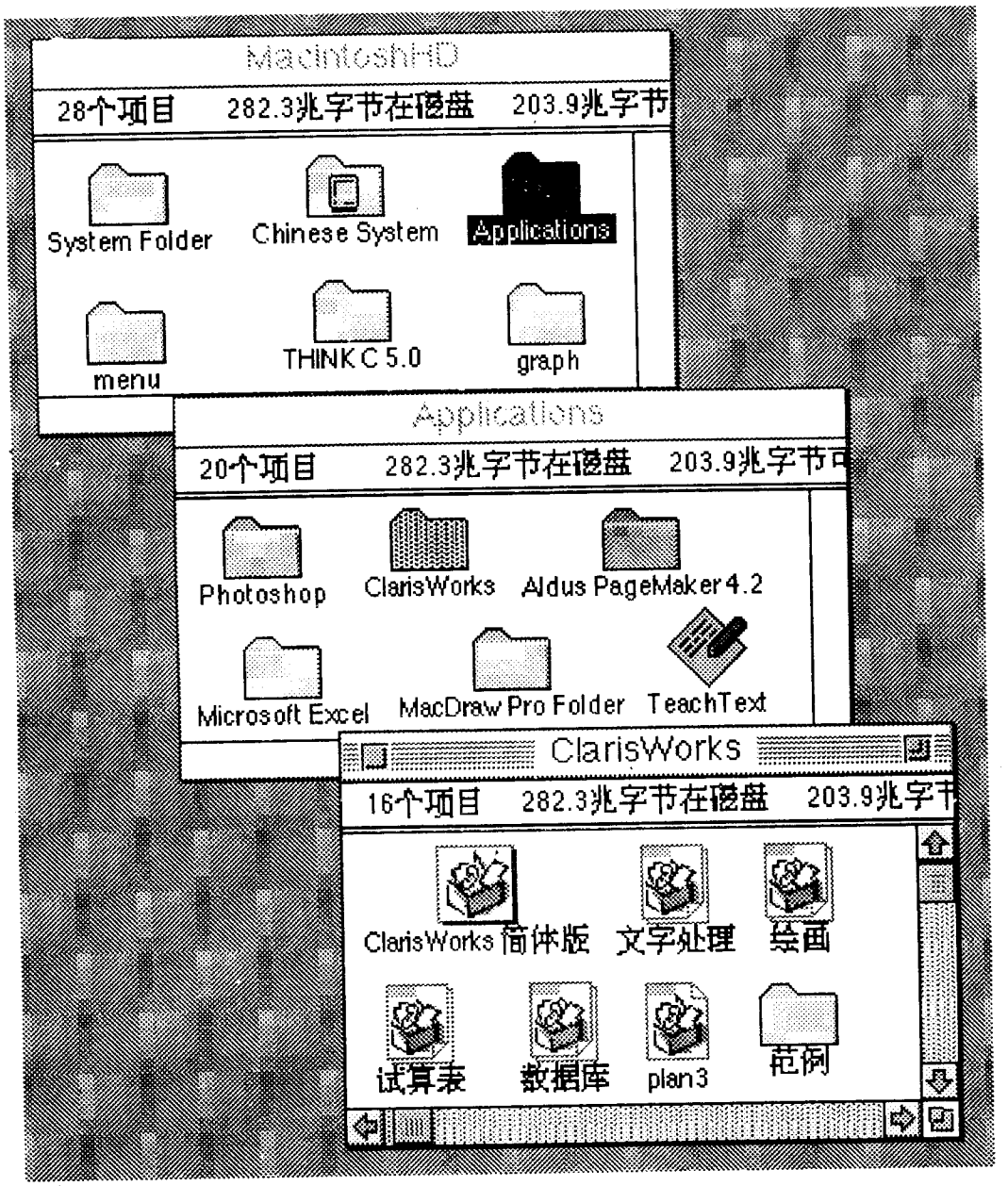

图 1-9 Macintosh HFS 示意图

系统7中用文件说明（file specification）来引用文件，解决了这一问题。文件说明的数据结构为FSSpec，定义如下：

```
Struct FSSpec
{
short vRefNum; /*卷号（Volume Reference ID）*/
long parID;  /*文件或子目录所在的目录号（Directory ID）*/
Str63 name;  /*指定的文件或子目录名*/
}
```

有很多系统调用，是通过文件说明来处理文件。例如将文件说明转换成工作目录号，以及相反的转换（如果你的应用程序要在旧的系统上运行，就需要这样转换）。

二、文件替身

可以看出，文件说明是文件的一个绝对参照，只要文件始终存放在同一位置，文件说明就能反复有效地提供文件的位置。但是，当文件移动位置以后会怎样呢？对于这种情况，系统7也有解决办法，这就是文件替身。

文件替身中含有一些查找信息，替身管理器（Alias Manager）可以用这些查找信息来寻找文件原身，于是应用程序便可用替身来跟踪文件的位置。当要使用某个文件时，只要将其替身送给替身管理器，便可找到文件。如果文件被移动了，替身管理器会修改查找信息以反映文件的当前位置。找到文件之后，应用程序便将替身转换成文件说明用来打开文件。

文件替身记录的结构定义如下：

```
Struct Alias Record
{
OsType userType;
/*由建立这一替身的应用程序使用，用于存放说明，或其它4字节信息。替身管理器创建这一记录时，其初值为0。*/
unsigned short aliasSize;
/*替身记录的总长度*/
/*其它可变长的查找信息*/
};
```

其中第二项记录总长度是必不可少的，一般为200到300字节。此项之后是由替身管理器使用的可变长数据，用于寻找目标文件。

要得到一个文件的替身，只要将文件处于当前位置时的文件说明送给替身管理器。以后，每次只要将替身回送给替身管理器，便可得到文件说明。如果文件被移动了，替身管理器便用查找信息来查找文件，并返回一个针对新地址的文件说明。替身很有用而且很方便，下面我们举例说明替身的使用。

假设你的某个文档中的数据，要与嵌在另一个文件中的某些数据连接，则关闭文档时，必须存储另一个文件的物理地址，以及你在其中查找数据的偏移地址。

替身正好可以用来存放所有这些信息，所以应将另一个文件的文件说明转换为替身。

然后，调用内存管理器将替身扩大2字节，用于存放数据在另一文件中的偏移地址，并将这个替身存入你的文档文件中。（注意：虽然你改变了替身的大小，但不要改变替身记录中的aliasSize值，因为aliasSize只是用来说明替身管理器所用的查找信息的长度。）

当你打开文档并读出替身后，可用替身来得到另一文件的说明。然后用替身记录中的aliasSize值找到查找信息的结束位置，其后便是所需数据在另一文件中的偏移。

1.6 Gestalt管理器

在使用替身、文件说明，或其它特性（如QuickTime电影）之前，必须确认操作系统支持这些特性。这就是Gestalt管理器的功能。

Gestalt管理器提供一个同名的函数可以准确得到系统的可用功能。这一调用有两个参数：selector和response。

Gestalt函数的类型及参数定义如下：

```
typedef unsigned long OSType;
puscal OSErr Gestalt(OSType selector, long *response);
```

selector是OSType类型，是一个无符号长整数，通过这个参数，你可以选择你需要哪个管理器的哪种信息。Selector的部分可选值如下：

环境选项：

```
#define     gestaltAlias MgrAttr      'alis'
#define     gestaltFSAttr             'fs '
#define     gestaltAppleEventsAttr    'event'
#define     gestaltEditionMgrAttr     'edtn'
#define     gestaltQuickTime          'qtim'
#define     gestaltFontMgrAttr        'font'
#define     gestaltKeyboardType       'kbd '
#define     gestaltAppleTalkVersion   'atlk'
#define     gestaltAUXVersion         'a/ux'
#define     gestaltConnMgrAttr        'conn'
#define     gestaltCRMAttr            'crm '
#define     gestaltCTBVersion         'ctbv'
#define     gestaltToolboxTable       'tbtt'
#define     gestaltDITLExtAttr        'ditl'
#define     gestaltScriptCount        'scr#'
```

信息选项：

```
#define     gestaltMachineIcon        'micn'
```